

Chapter 6

순차회로 모델링

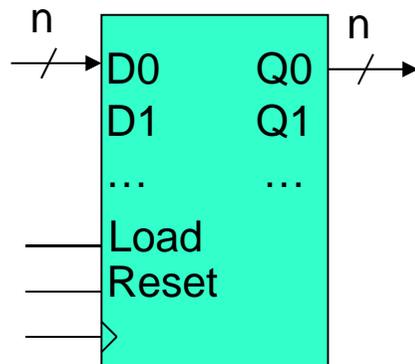
6.1 Registers

- Registers

- n-bit data를 저장하는 기억소자

- Load 제어신호가 있는 register

- Reset=0일 때에 0을 저장 (비동기)
- clock의 positive edge에서 Load=1이면 입력 data를 저장



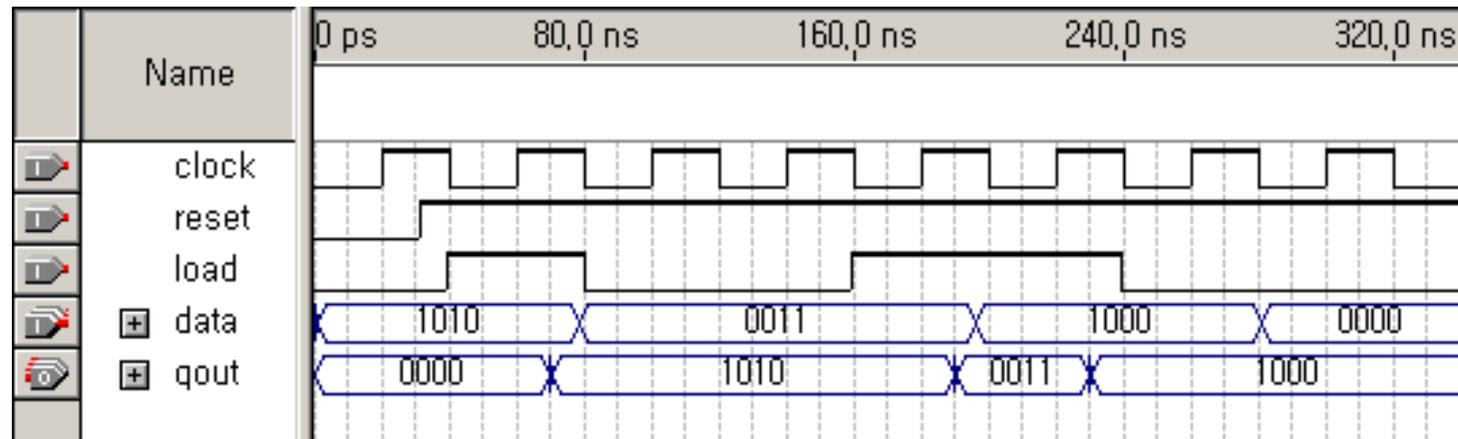
Reset	Load	Qi
0	x	0
1	1	Di
1	0	Qi

Reset은 비동기 동작, active low
Load는 동기 동작, active high

Load 제어신호가 있는 register

```
module register (qout, data, load, reset, clock);
    output [3:0] qout;
    input [3:0] data;
    input load, reset, clock;
    reg [3:0] qout;

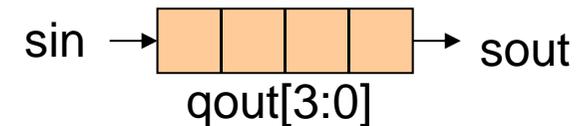
    always @ (posedge clock or negedge reset) begin
        if (~reset) qout <= 4'b0;           // reset
        else if (load) qout <= data;       // load
                                           // 나머지는 hold
    end
endmodule
```



Shift Register

■ 4-bit shift register

- clock의 positive edge에서 한자리씩 shift (right)
- reset=0일 때에 0을 저장 (비동기)
- 직렬 입출력



```
module shiftreg(sout, sin, reset, clock);
    output  sout;                // serial out
    input   sin, reset, clock;   // serial in ...
    reg [3: 0] qout;

    assign  sout = qout[0];
    always @ (negedge reset or posedge clock) begin
        if (reset == 1'b0)
            qout <= 4'b0;        // reset
        else begin
            qout[3] <= sin;      // shift right
            qout[2] <= qout[3];
            qout[1] <= qout[2];
            qout[0] <= qout[1];
        end
    end
end
endmodule
```



Shift Register (계속)

- 다른 표현 - vector 자료형 이용

```
module shiftreg(sout, sin, reset, clock);
    output  sout;                // serial out
    input   sin, reset, clock;   // serial in ...
    reg [3: 0] qout;

    assign  sout = qout[0];
    always @ (negedge reset or posedge clock) begin
        if (reset == 1'b0) qout <= 4'b0;    // reset
        else qout <= { sin, qout[3:1] };     // shift left
    end
endmodule
```

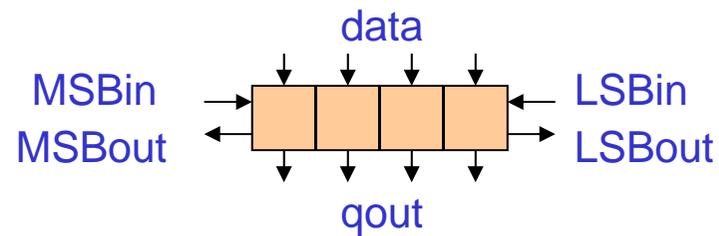


양방향 Shift Register

■ 동작

- 병렬 load, 양방향 shift 기능
(선택신호 s[1:0]로 기능선택)
- 직렬/병렬 입출력
- 동기 reset (active low)

s[1:0]	operation
0 0	hold(변화없음)
0 1	shift right
1 0	shift left
1 1	load



양방향 Shift Register (계속)

```
module bishiftreg(qout, MSBout, LSBout, data, MSBin, LSBin, sel, reset, clock);
    output [3: 0] qout;      // parallel out
    output MSBout, LSBout;  // serial out
    input [3:0] data;       // parallel in
    input MSBin, LSBin;     // serial in
    input [1:0] sel;        // function select
    input clock, reset;
    reg [3:0] qout;

    assign MSBout = qout[3];
    assign LSBout = qout[0];
    always @(posedge clock) begin
        if (reset==0) qout <= 0;
        else begin
            case (sel)
                0: qout <= qout;           // hold
                1: qout <= {MSBin, qout[3:1]}; // shift right
                2: qout <= {qout[2:0], LSBin}; // shift left
                3: qout <= data;           // parallel Load
            endcase
        end
    end
end
endmodule
```

불필요



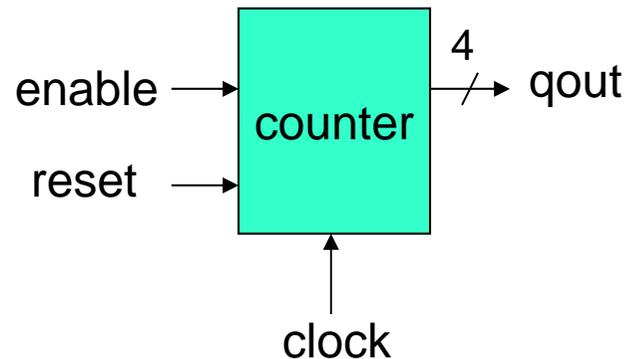
6.2 Counters

■ counter

- 미리 정해진 순서로 상태가 변하는 순차회로
- 대개 증가 또는 감소 순서대로 변화

■ 4비트 동기식 2진카운터

- enable=1인 동안 매 clock마다 다음 순서로 변화
0000 → 0001 → 0010 → ... → 1110 → 1111 → 0000
- 비동기 reset



동기식 2진 카운터

■ 덧셈 연산자를 사용한 설계

```
module counter(qout, enable, reset, clock);
    output [3:0] qout;
    input enable, reset, clock;
    reg [3:0] qout;

    always @(negedge reset or posedge clock) begin
        if (~reset) qout <= 0;
        else if (enable) qout <= qout + 1;
    end
endmodule
```

- adder 라이브러리를 사용하여 합성될 수도 있음
- counter 라이브러리를 사용하여 합성될 수도 있음



동기식 2진 카운터 (계속)

- 덧셈연산자를 사용하지 않은 설계

```
module counter2(qout, enable, reset, clock);
    output [3:0] qout;
    input enable, reset, clock;
    reg [3:0] qout;

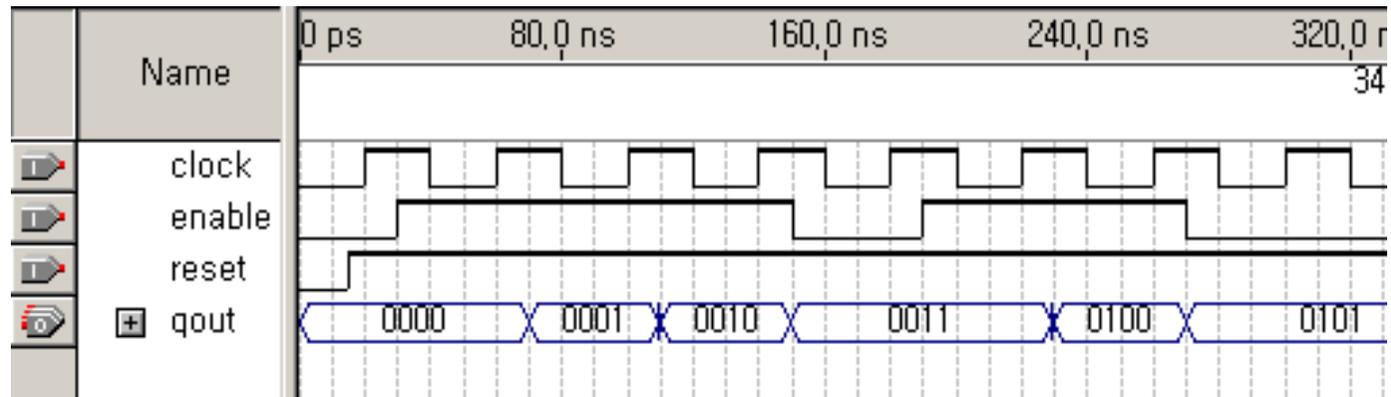
    always @(negedge reset or posedge clock) begin
        if (~reset) qout <= 0;
        else if (enable) begin
            qout[0] <= ~qout[0];
            if (qout[0] == 1'b1) qout[1] <= ~qout[1];
            if (qout[1:0] == 2'b11) qout[2] <= ~qout[2];
            if (qout[2:0] == 3'b111) qout[3] <= ~qout[3];
        end
    end
endmodule
```

- 아래자리들이 모두 1일 때 윗자리가 바뀜
- 설계된 대로 합성됨



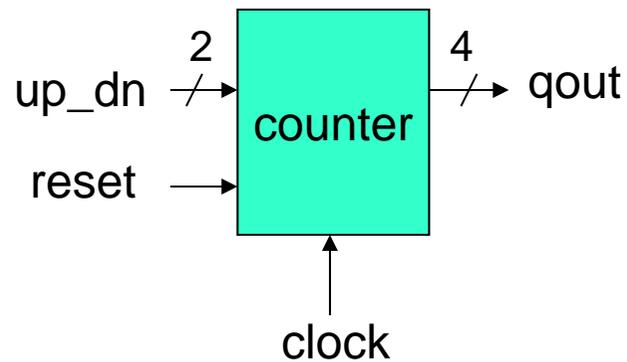
동기식 2진 카운터 (계속)

■ Timing도



Up-Down 카운터

- 4비트 up-down 카운터
 - 증가 또는 감소 동작 선택 가능



reset	up_dn	operation
0	xx	reset (clear)
1	01	up (increment)
1	10	down(decrement)
1	00,11	hold (no change)

Up-Down 카운터 (계속)

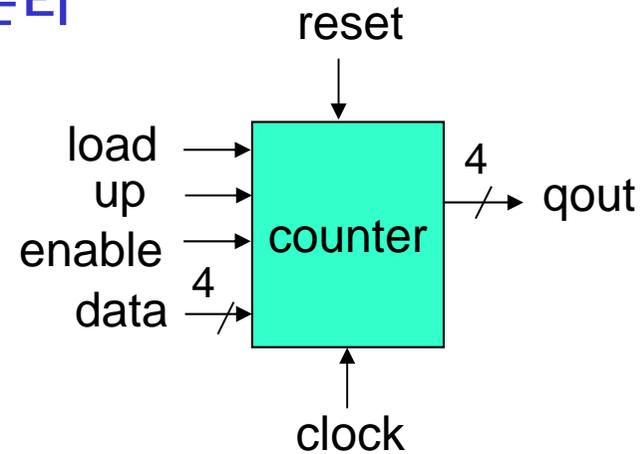
```
module updown_counter(qout, up_dn, reset, clock);
    output [3:0] qout;
    input [1:0] up_dn;
    input reset, clock;
    reg [3:0] qout;

    always @ (negedge clock or negedge reset) begin
        if (reset == 0) qout <= 4'b0;           // reset
        else if (up_dn == 2'b01) qout <= qout + 1; // up
        else if (up_dn == 2'b10) qout <= qout - 1; // down
    end
endmodule
```



병렬로드 Up-Down 카운터

■ 4비트 병렬로드 up-down 카운터



reset	load	enable	up	operation
1	x	x	x	reset (clear)
0	1	x	x	parallel load
0	0	1	1	up (increment)
0	0	1	0	down(decrement)
0	0	0	x	hold (no change)

비동기

병렬로드 Up-Down 카운터 (계속)

```
module updown_counter2(qout, data, load, up, enable, reset, clock);
    output [3:0] qout;
    input load, up, enable, reset, clock;
    input [3:0] data;
    reg [3:0] qout;

    always @(posedge reset or posedge clock) begin
        if (reset) qout <= 4'b0;           // reset
        else if (load) qout <= data;      // load
        else if (enable) begin           // enable
            if (up) qout <= qout + 1;    // up
            else qout <= qout - 1;       // down
        end
    end
end
endmodule
```



modulo-N 카운터

- modulo-N 카운터
 - N개의 상태를 반복하여 변하는 카운터 (대개 0부터 N-1까지)
- modulo-10 카운터 (동기식 reset)
 - 직접 설계

```
module counter10(qout, enable, reset, clock);
    output [3:0] qout;
    input enable, reset, clock;
    reg [3:0] qout;

    always @(posedge clock) begin
        if (~reset) qout <= 0;
        else if (enable) begin
            if (qout==9) qout <= 0;
            else qout <= qout + 1;
        end
    end
end
endmodule
```



modulo-N 카운터

```
// modulo-10 counter
module counter_10(qout, enable, reset, clock);
    output [3:0] qout;
    input enable, reset, clock;
    wire rst;

    counter_s u1 (qout, enable, ~rst, clock);

    assign rst = ~reset || (qout == 9);

endmodule

// counter with synchronous reset
module counter_s(qout, enable, reset, clock);
    output [3:0] qout;
    input enable, reset, clock;
    reg [3:0] qout;

    always @(posedge clock) begin
        if (~reset) qout <= 0;
        else if (enable) qout <= qout + 1;
    end
endmodule
```

이진카운터를
사용한 설계
(동기식reset 이용)



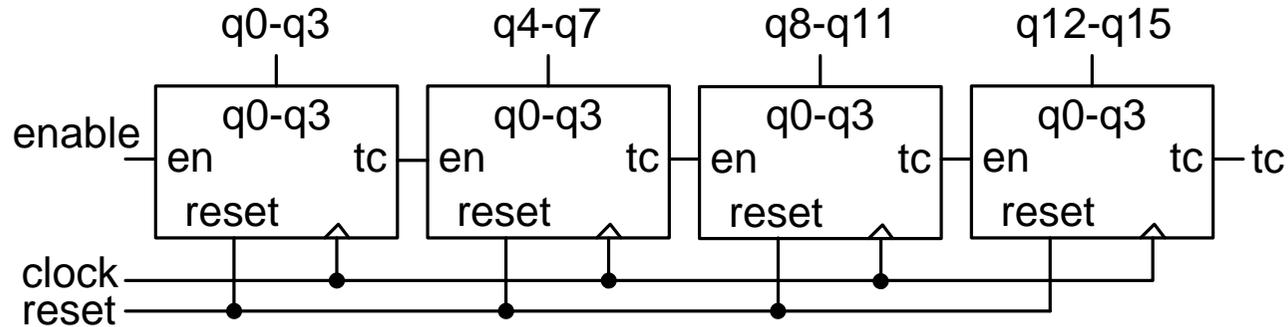
캐리 출력을 갖는 4비트 이진 카운터

- carry output (또는 terminal count)
 - 카운터의 마지막 상태에서(enable이 1일 때에) 1이 됨
 - 카운터를 직렬 연결하여 큰 카운터를 구성할 때 상위 카운터의 enable 신호로 사용됨

```
module counter4(qout, tc, enable, reset, clock);
    output [3:0] qout;
    output tc;
    input enable, reset, clock;
    reg [3:0] qout;

    always @(posedge clock) begin
        if (~reset) qout <= 0;
        else if (enable) qout <= qout + 1;
    end
    assign tc = (qout == 4'b1111) & enable;
endmodule
```

4비트 이진카운터를 사용한 16비트 카운터



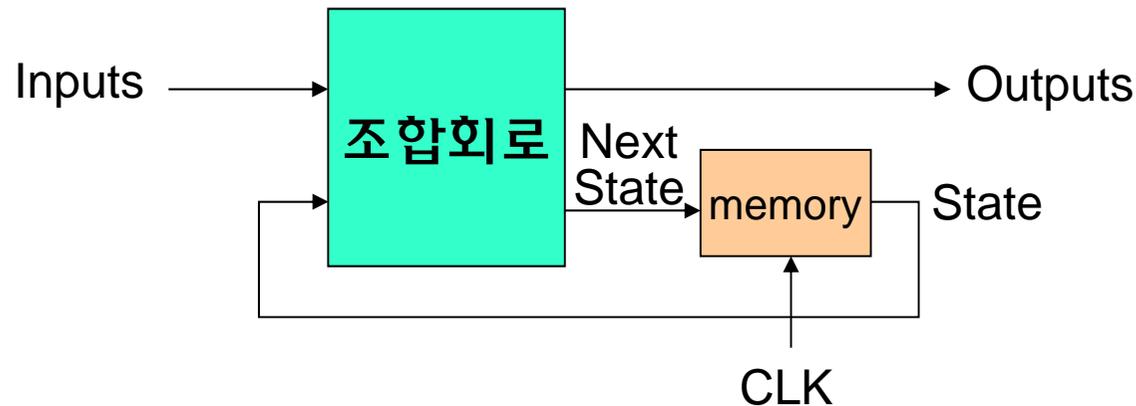
```
module counter16(qout, tc, enable, reset, clock);
    output [15:0] qout;
    output tc;
    input enable, reset, clock;
    wire enable1, enable2, enable3;

    counter4 u1 (qout[3:0],enable1, enable, reset, clock);
    counter4 u2 (qout[7:4],enable2,enable1, reset, clock);
    counter4 u3 (qout[11:8],enable3,enable2, reset, clock);
    counter4 u4 (qout[15:12],tc,enable3, reset, clock);
endmodule
```



6.3 순차회로와 유한상태기계(FSM)

■ 순차회로의 구조



- 순차회로 표현: 두 함수를 사용
 - state transition function: $\text{Next State} = f(\text{Inputs}, \text{State})$
 - output function: $\text{Output} = g(\text{Inputs}, \text{State})$
- Next State가 다음 클럭에 State로 저장됨 → state transition

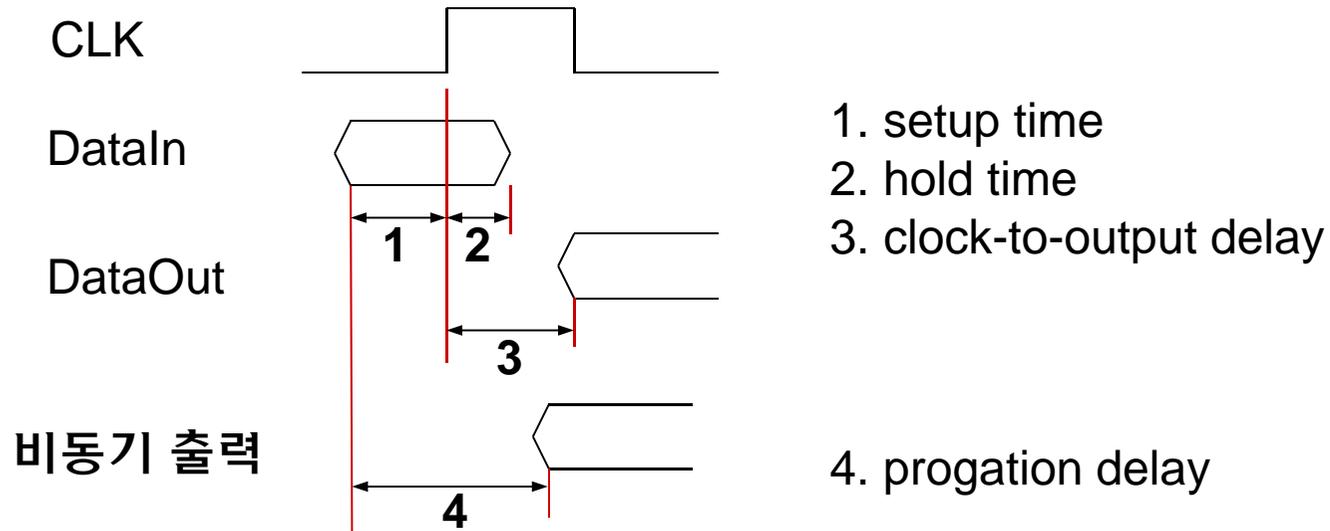
Timing

■ Setup & Hold Time

- Clock에 동기되어 동작하는 입력이 Clock의 edge 전/후에 안정되게 입력되어야 하는 최소 시간

■ Propagation Delay & Clock-to-Output Delay

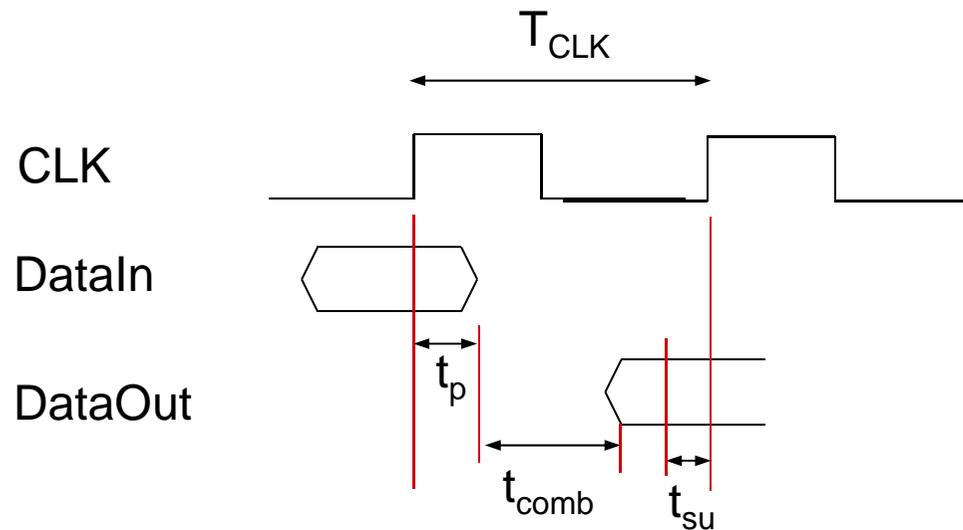
- 회로의 입력의 변화에 대해서 출력이 변화되는 최대 시간
- Clock에 동기되어 변하는 출력은 Clock edge에서 출력까지의 최대 지연 시간으로 나타냄



조합회로 delay와 Clock 주기

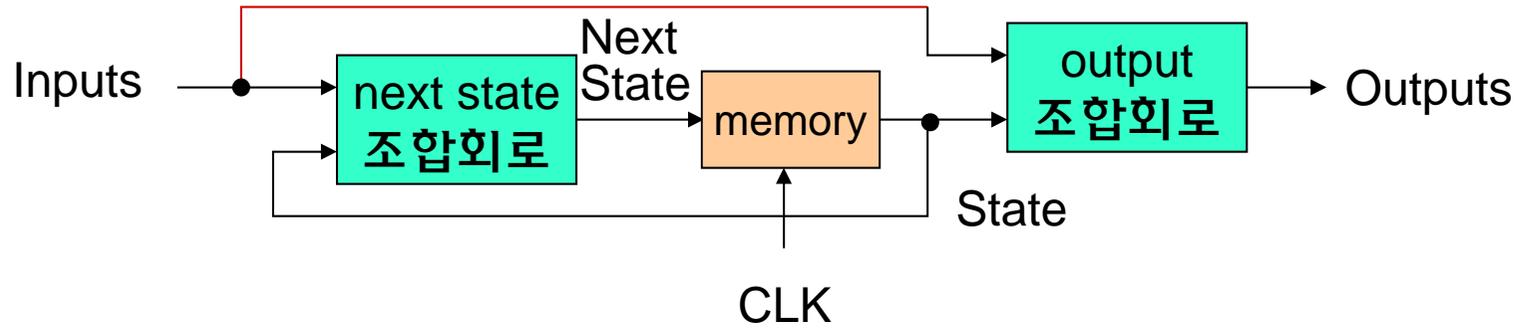
- 순차회로의 정상 동작을 위해서는 조합 회로 delay(플립플롭 지연시간, 셋업시간 포함) 는 clock 주기보다 작아야 함.

$$T_{\text{CLK}} > t_p + t_{\text{comb}} + T_{\text{su}}$$

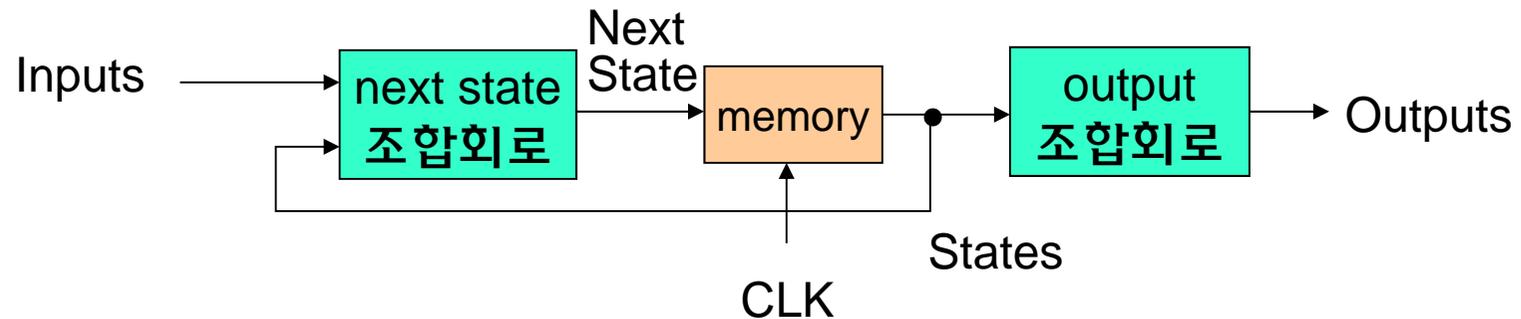


Mealy machine과 Moore machine

- Mealy machine: Output = f(Inputs, States)

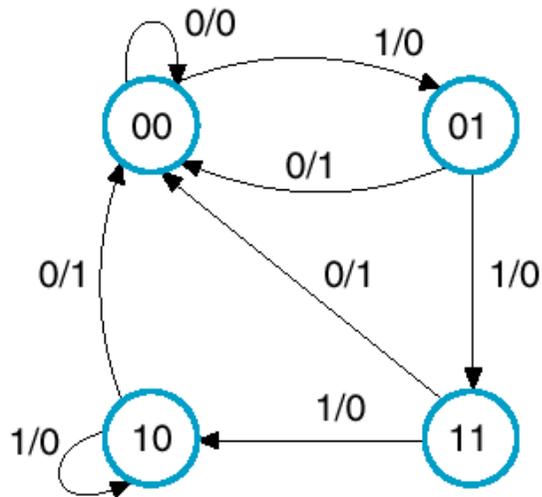


- Moore machine: Output = f(States) ... CLK에 동기화된 출력
(States가 CLK에 동기화된 신호임)

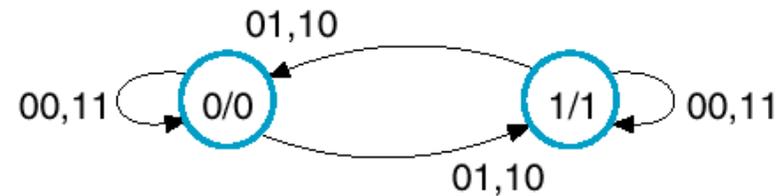
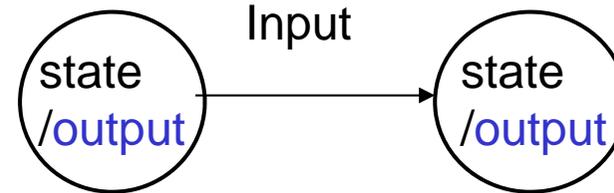


State Diagram

Mealy Machine



Moore Machine



State Transition Table

Present State		Input	Next State		Output
A	B	X	A	B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0



상태도의 Verilog 설계(1)

■ 방법1 :

- 클럭에 동기되어 next state가 state로 저장되는 register 회로 기술
- next state와 output 출력 제공 조합회로를 같은 case문에 함께 기술

```
module mealy1(out, state, in, reset, clock);  
    output out;  
    output [1:0] state;  
    input in, reset, clock;  
    reg out;  
    reg [1:0] state, nextstate;
```

```
    always @(posedge clock or negedge reset) begin  
        if (~reset) state <= 2'b00;  
        else state <= nextstate;  
    end
```

상태도의 Verilog 설계(1) - 계속

```
always @(state or in) begin
    out = 0; —————> default output값
    case (state)
        2'b00: if (in==0) nextstate = 2'b00;
                else nextstate = 2'b01;
        2'b01: if (in==0) begin nextstate = 2'b00; out = 1; end
                else nextstate = 2'b11;
        2'b10: if (in==0) begin nextstate = 2'b00; out = 1; end
                else nextstate = 2'b10;
        2'b11: if (in==0) begin nextstate = 2'b00; out = 1; end
                else nextstate = 2'b10;
        default: nextstate = 2'bxx;
    endcase
end
endmodule
```



상태도의 Verilog 설계(2)

■ 방법2 : 1-bit output에 대해서

- 클럭에 동기되어 next state가 state로 저장되는 회로 기술(방법1과 동일)
- next state 회로를 case문으로, output 회로를 assign 문으로 기술

```
always @(state or in) begin
```

```
  case (state)
```

```
    2'b00: if (in==0) nextstate = 2'b00;
```

```
          else nextstate = 2'b01;
```

```
    2'b01: if (in==0) nextstate = 2'b00;
```

```
          else nextstate = 2'b11;
```

```
    2'b10: if (in==0) nextstate = 2'b00;
```

```
          else nextstate = 2'b10;
```

```
    2'b11: if (in==0) nextstate = 2'b00;
```

```
          else nextstate = 2'b10;
```

```
    default: nextstate = 2'bxx;
```

```
  endcase
```

```
end
```

```
assign out = (state==2'b01)&&(in==0) || (state==2'b11) && (in==0) ||  
             (state==2'b10)&&(in==0);
```

```
endmodule
```



상태도의 Verilog 설계(3)

■ 방법3

- nextstate 조합회로를 별도로 구현하지 않고 CLK에 동기되어 동작하는 state 레지스터 회로에 포함하여 case문을 사용하여 설계
- output 출력 조합회로를 case 또는 assign 문을 사용하여 설계

```
reg [1:0] state; —————> nextstate 없음
```

```
always @(posedge clock or negedge reset) begin
```

```
  if (~reset) state <= 2'b00;
```

```
  else begin
```

```
    case (state)
```

```
      2'b00: if (in==1) state <= 2'b01;
```

```
      2'b01: if (in==0) state <= 2'b00;
```

```
            else state <= 2'b11;
```

```
      2'b10: if (in==0) state <= 2'b00;
```

```
      2'b11: if (in==0) state <= 2'b00;
```

```
            else state <= 2'b10;
```

```
      default: state <= 2'b00;
```

```
    endcase
```

```
  end
```

```
end
```

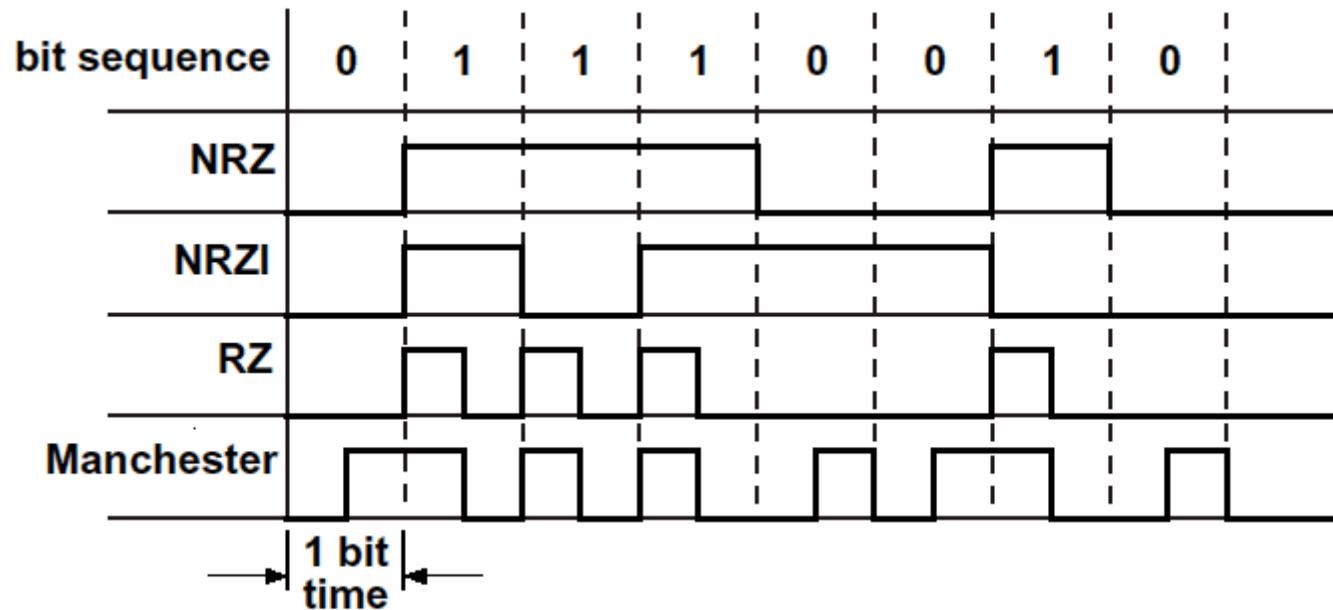
state변화가 없는
경우에는 기술하지
않음



6.4 (예) 맨체스터코드 변환기

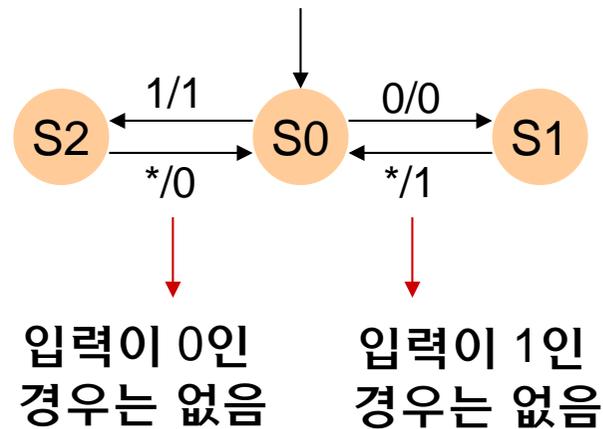
■ Line code formats

- NRZ (non-return-to-zero) : 비트 값을 그대로 전송
- NRZI (non-return-to-zero invert-on-ones)
- RZ (return-to-zero)
- Manchester: 0→1 for 0, 1→0 for 1



NRZ-to-Manchester Code Converter - Mealy

■ State Diagram – Mealy machine



Manchester코드 생성회로의 CLK 주파수는
NRZ코드 생성회로의 CLK주파수의 2배이어야 함

Verilog Model – output function을 state transition function과 함께 기술

```
// NRZ to Manchester Converter - Mealy machine
module Manchester1(out, in, reset, clock, state);
    output out;
    output [1:0] state;
    input in, reset, clock;
    reg [1:0] state, next_state;
    reg out;
    parameter S0 = 0, S1 = 1, S2 = 2;

    always @(posedge clock or negedge reset) begin
        if (reset==0) state <= S0;        // initial state
        else state <= next_state;
    end

    // next state & output combinational logic
    always @(state or in) begin
        out = 0;    // default output
        case (state)
            S0: if (in == 1'b0) next_state = S1;
                else if (in == 1'b1) begin next_state = S2; out = 1; end
                else begin next_state = 2'bx; out = 1'bx; end
            S1: begin next_state = S0; out = 1; end
            S2: begin next_state = S0; end
            default: begin next_state = 2'bx; out = 1'bx; end
        endcase
    end
endmodule
```



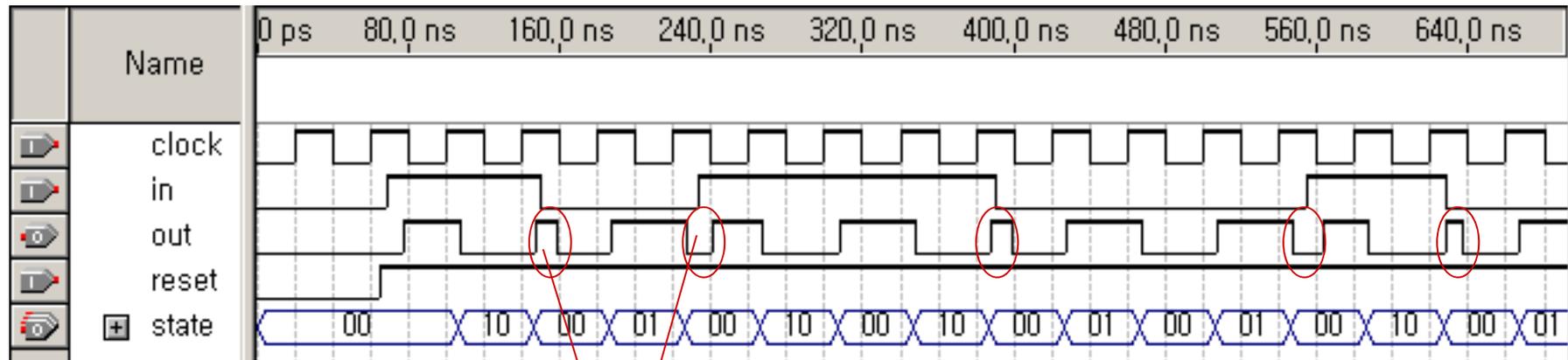
Verilog Model – output function을 assign문으로 기술

```
// NRZ to Manchester Converter - Another Mealy machine
module Manchester2(out, in, reset, clock, state);
    output out;
    output [1:0] state;
    input in, reset, clock;
    reg [1:0] state, next_state;
    parameter S0 = 0, S1 = 1, S2 = 2;

    always @(posedge clock or negedge reset) begin
        if (reset==0) state <= S0;        // initial state
        else state <= next_state;
    end
    // next state combinational logic
    always @(state or in) begin
        case (state)
            S0: if (in == 0) next_state = S1;
                else if (in == 1) next_state = S2;
                else next_state = 2'bx;
            S1: next_state = S0;
            S2: next_state = S0;
            default: next_state = 2'bx;
        endcase
    end
    // output combinational logic
    assign out = (state==S0) && (in==1) || (state==S1) && (in==0);
endmodule
```



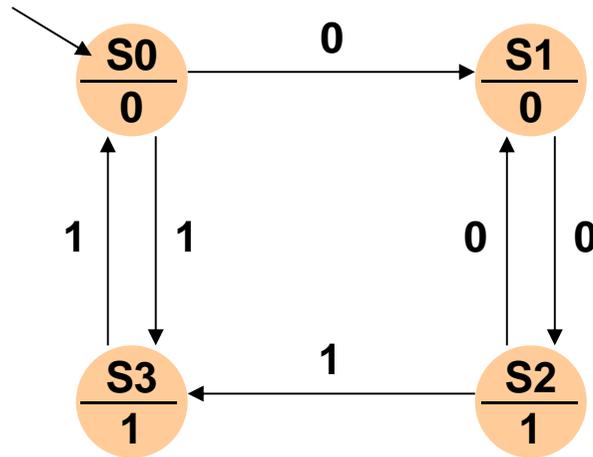
■ Timing 도



입력변화와 클럭변화의
불일치에 의한 glitch 발생

NRZ-to-Manchester Code Converter - Moore

■ State Diagram – Moore Machine



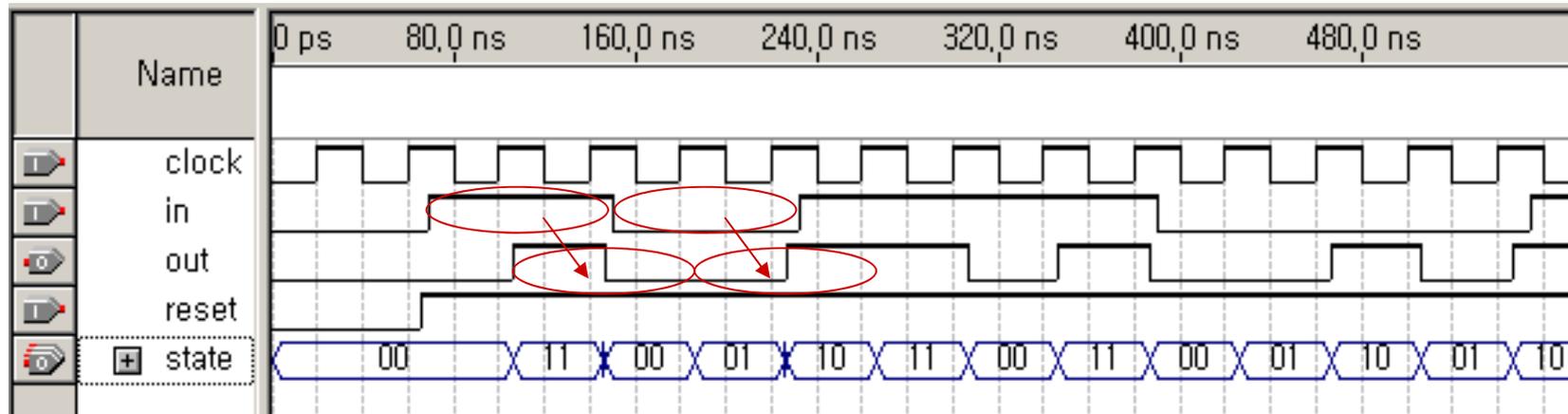
Verilog Model – output function을 assign문으로 기술

```
module Manchester_Moore(out, in, reset, clock, state);
    output out;
    output [1:0] state;
    input in, reset, clock;
    reg [1:0] state, next_state;
    parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3;

    always @(posedge clock or negedge reset) begin
        if (reset==0) state <= S0;        // initial state
        else state <= next_state;
    end
    // next state combinational logic
    always @(state or in) begin
        case (state)
            S0, S2: if (in == 0) next_state = S1;
                   else next_state = S3;
            S1: next_state = S2;
            S3: next_state = S0;
            default: next_state = 2'bx;
        endcase
    end
    // output combinational logic
    assign out = (state==S2) || (state==S3);
endmodule
```



■ Timing도

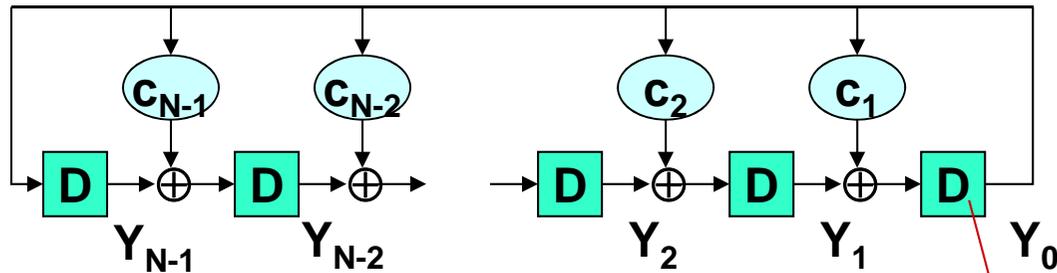


- 출력 code에 glitch가 발생하지 않음 ← 클럭에 동기되어 출력
- NRZ코드 입력(in)보다 1 CLK 후에 Manchester code 출력(out)이 생성됨

6.5 Linear-Feedback Shift Register

Linear Feedback Shift Register(LFSR)

- data compression에 사용
- cyclic redundancy code(CRC) 생성에 사용
- pseudo random pattern generator로 사용



$$Y[k-1] = (c[k] Y[0]) \oplus Y[k] \quad (k=1 \dots N-1)$$

$$Y[N-1] = Y[0]$$

if (c[k]=1) $Y[k-1] = Y[0] \oplus Y[k]$
 if (c[k]=0) $Y[k-1] = Y[k]$

1 clock delay
D flip-flop으로 구현



LFSR model

■ 8-bit LFSR

```
module LFSR(Y, reset, clock);
    parameter initial_state = 8'b1001_0001; // 91h
    parameter [7:1] C = 7'b100_1111; // coefficient
    input reset, clock;
    output [7:0] Y;
    reg [7:0] Y;

    always @ (posedge clock) begin
        if (!reset) // Active-low reset to initial state
            Y <= initial_state;
        else begin
            Y[7] <= Y[0];
            Y[6] <= C[7] ? Y[7] ^ Y[0] : Y[7];
            Y[5] <= C[6] ? Y[6] ^ Y[0] : Y[6];
            Y[4] <= C[5] ? Y[5] ^ Y[0] : Y[5];
            Y[3] <= C[4] ? Y[4] ^ Y[0] : Y[4];
            Y[2] <= C[3] ? Y[3] ^ Y[0] : Y[3];
            Y[1] <= C[2] ? Y[2] ^ Y[0] : Y[2];
            Y[0] <= C[1] ? Y[1] ^ Y[0] : Y[1];
        end
    end
endmodule
```

} 비슷한 동작이
반복됨



LFSR model – 반복문 사용

```
module LFSR2(Y, reset, clock);
  parameter N = 8; // length
  parameter initial_state = 8'b1001_0001; // 91h
  parameter [N-1:1] C = 7'b100_1111; // coefficient
  input reset, clock;
  output [N-1:0] Y;
  reg [N-1:0] Y;
  integer k;

  always @ (posedge clock) begin
    if (!reset) // Active-low reset to initial state
      Y <= initial_state;
    else begin
      Y[N-1] <= Y[0];
      for (k = 1; k <= N-1; k=k+1)
        Y[k-1] <= C[k] ? Y[k] ^ Y[0] : Y[k];
    end
  end
endmodule
```

→ 하드웨어 신호 모델링이 아님

→ for 반복문



LFSR 응용: CRC generator

- serial CRC generator

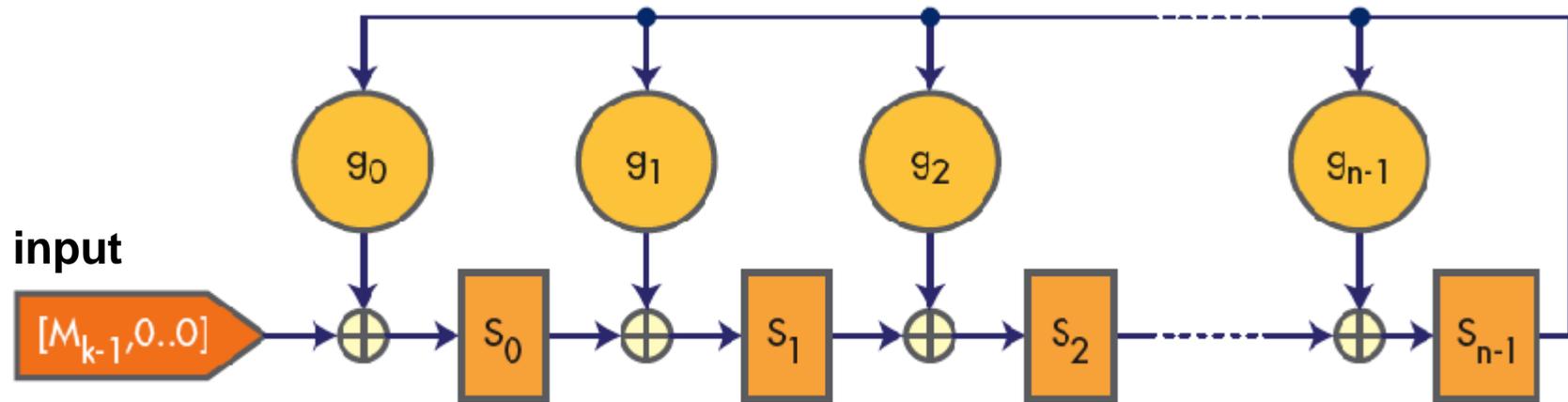


Figure 1

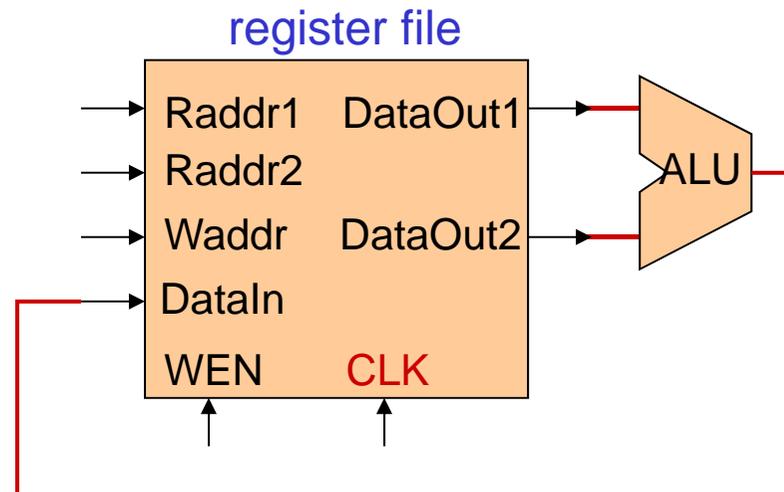
$$C_{n-1}(X) = [S_{n-1} \dots S_1 S_0], \text{ after } k + n \text{ clocks}$$

k-bit message + n-bit zero padding 입력 → n-bit CRC code 생성

6.6 Register File

■ register file

- 적은 수의 register들로 구성되어 있는 순차회로
- register에 대한 read, write 동작을 동시에 수행
- 두 개의 output을 갖는 경우가 많음 (두 register 동시에 read 가능)

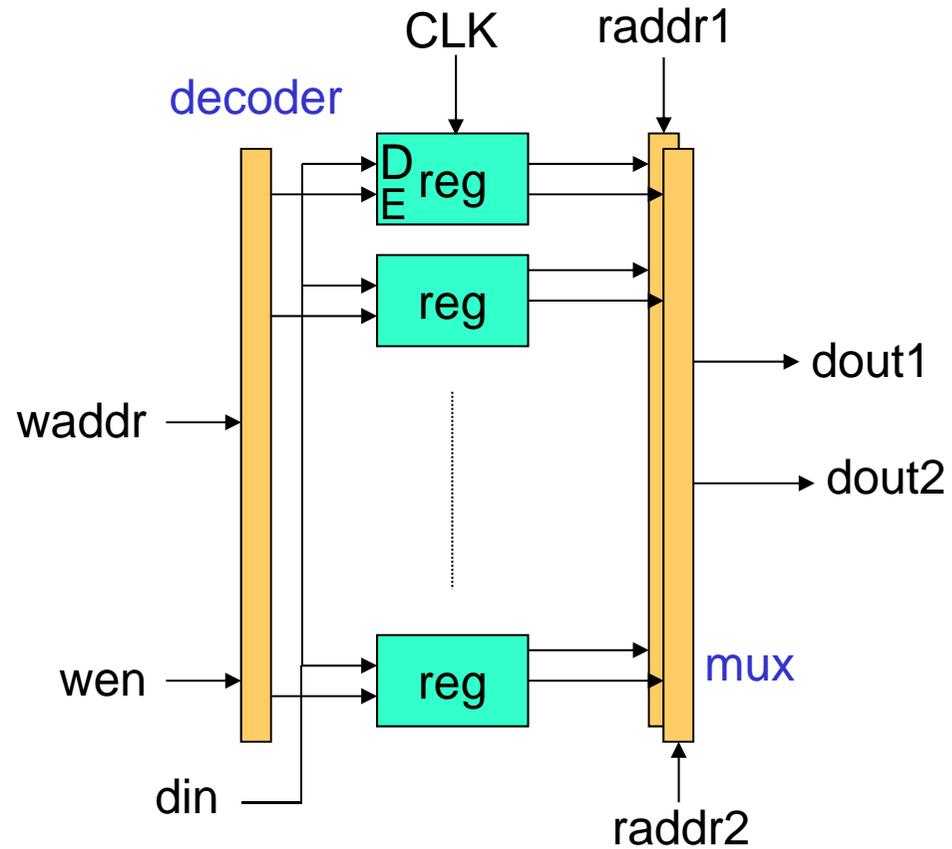


- WEN=1이면 지정된(Waddr) write register에 clock에 동기되어 데이터가 저장됨
- 지정된(Raddr1,2) read register에 저장된 값을 data output으로 출력



Register File (계속)

■ 내부 구성도



Register File 설계 (1)

```
module regfile(dout1, dout2, din, raddr1, raddr2, waddr, wen, clock);
    output [31:0] dout1, dout2;
    input [31:0] din;
    input [3:0] raddr1, raddr2, waddr;
    input wen, clock;

    reg [31:0] reg_file [0:15];    // 32 bit x 16 word memory declaration
    reg [31:0] dout1, dout2;
    integer i;

    // write
    always @(posedge clock) begin
        for (i=0; i<16; i=i+1) begin
            if (wen & (waddr==i)) reg_file[i] <= din;
        end
    end

    // read
    always @(raddr1 or raddr2 or reg_file) begin
        dout1 = 32'bx;
        dout2 = 32'bx;
        for (i=0; i<16; i=i+1) begin
            if (raddr1==i) dout1 = reg_file[i];
            if (raddr2==i) dout2 = reg_file[i];
        end
    end
endmodule
```



Register File 설계 (2)

■ Verilog model

```
module register_file(dout1, dout2, din, raddr1, raddr2, waddr, wen, clock);
    output [31:0] dout1, dout2;
    input [31:0] din;
    input [3:0] raddr1, raddr2, waddr;
    input wen, clock;
    reg [31:0] reg_file [0:15];    // 32 bit x 16 word memory declaration

    assign dout1 = reg_file[raddr1];    // read
    assign dout2 = reg_file[raddr2];
    always @ (posedge clock) begin    // write
        if (wen) reg_file[waddr] <= din;
    end
endmodule
```



Register File timing도

■ Timing도

